Javascript

# Variables

Data is anything that is meaningful to the computer.

JavaScript provides eight different data types which are undefined, null, boolean, string, symbol, bigint, number, and object.

For example, computers distinguish between numbers, such as the number 12, and strings, such as "12", "dog", or "123 cats", which are collections of characters. Computers can perform mathematical operations on a number, but not on a string.

Variables allow computers to store and manipulate data in a dynamic fashion. They do this by using a "label" to point to the data rather than using the data itself. Any of the eight data types may be stored in a variable.

We tell JavaScript to create or declare a variable by putting the keyword var in front of it, like so:

var ourName;

creates a variable called ourName. In JavaScript we end statements with semicolons. Variable names can be made up of numbers, letters, and $ or \_, but may not contain spaces or start with a number.

# Storing Values with the assignment operator

In JavaScript, you can store a value in a variable with the assignment operator (=).

myVariable = 5;

This assigns the Number value 5 to myVariable.

If there are any calculations to the right of the = operator, those are performed before the value is assigned to the variable on the left of the operator.

var myVar;

myVar = 5;

First, this code creates a variable named myVar. Then, the code assigns 5 to myVar.

Now, if myVar appears again in the code, the program will treat it as if it is 5.

# Initializing Variables with the Assignment Operator

It is common to initialize a variable to an initial value in the same line as it is declared.

var myVar = 0;

Creates a new variable called myVar and assigns it an initial value of 0.

Define a variable a with var and initialize it to a value of 9.

# Declare String Variables

Previously you used the following code to declare a variable:

var myName;

But you can also declare a string variable like this:

var myName = "your name";

"your name" is called a string literal. A string literal, or string, is a series of zero or more characters enclosed in single or double quotes.

# Understanding Uninitialized Variables

When JavaScript variables are declared, they have an initial value of undefined.

If you do a mathematical operation on an undefined variable your result will be NaN which means "Not a Number".

If you concatenate a string with an undefined variable, you will get a string of undefined.

# Understanding Case Sensitivity in Variables

In JavaScript all variables and function names are case sensitive. This means that capitalization matters.

MYVAR is not the same as MyVar nor myvar. It is possible to have multiple distinct variables with the same name but different casing. It is strongly recommended that for the sake of clarity, you do not use this language feature.

**Best Practice**

Write variable names in JavaScript in camelCase. In camelCase, multi-word variable names have the first word in lowercase and the first letter of each subsequent word is capitalized.

Examples:

var someVariable;

var anotherVariableName;

var thisVariableNameIsSoLong;

# Explore Differences Between the var and let Keywords

One of the biggest problems with declaring variables with the var keyword is that you can easily overwrite variable declarations:

var camper = "James";

var camper = "David";

console.log(camper);

In the code above, the camper variable is originally declared as James, and is then overridden to be David. The console then displays the string David.

In a small application, you might not run into this type of problem. But as your codebase becomes larger, you might accidentally overwrite a variable that you did not intend to. Because this behaviour does not throw an error, searching for and fixing bugs becomes more difficult.

A keyword called let was introduced in ES6, a major update to JavaScript, to solve this potential issue with the var keyword. You'll learn about other ES6 features in later challenges.

If you replace var with let in the code above, it results in an error:

let camper = "James";

let camper = "David";

The error can be seen in your browser console.

So unlike var, when you use let, a variable with the same name can only be declared once.

# Declare a Read-Only Variable with the const Keyword

The keyword let is not the only new way to declare variables. In ES6, you can also declare variables using the const keyword.

const has all the awesome features that let has, with the added bonus that variables declared using const are read-only. They are a constant value, which means that once a variable is assigned with const, it cannot be reassigned:

const FAV\_PET = "Cats";

FAV\_PET = "Dogs";

The console will display an error due to reassigning the value of FAV\_PET.

You should always name variables you don't want to reassign using the const keyword.

This helps when you accidentally attempt to reassign a variable that is meant to stay constant.

Note: It is common for developers to use uppercase variable identifiers for immutable values and lowercase or camelCase for mutable values (objects and arrays).

You will learn more about objects, arrays, and immutable and mutable values in later challenges.

Also in later challenges, you will see examples of uppercase, lowercase, or camelCase variable identifiers.

# Add Two Numbers with JavaScript

Number is a data type in JavaScript which represents numeric data.

Now let's try to add two numbers using JavaScript.

JavaScript uses the + symbol as an addition operator when placed between two numbers.

Example:

const myVar = 5 + 10;

myVar now has the value 15.

# Subtract One Number from Another with JavaScript

We can also subtract one number from another.

JavaScript uses the - symbol for subtraction.

Example

const myVar = 12 - 6;

myVar would have the value 6.

# Multiply Two Numbers with JavaScript

We can also multiply one number by another.

JavaScript uses the \* symbol for multiplication of two numbers.

Example:

const myVar = 13 \* 13;

myVar would have the value 169.

# Divide One Number by Another with JavaScript

We can also divide one number by another.

JavaScript uses the / symbol for division.

Example

const myVar = 16 / 2;

myVar now has the value 8.

# Increment a Number with JavaScript

You can easily increment or add one to a variable with the ++ operator.

i++;

is the equivalent of

i = i + 1;

Note: The entire line becomes i++;, eliminating the need for the equal sign.

# Decrement a Number with JavaScript

You can easily decrement or decrease a variable by one with the -- operator.

i--; is the equivalent of

i = i - 1;

Note: The entire line becomes i--;, eliminating the need for the equal sign.

# Create Decimal Numbers with JavaScript

We can store decimal numbers in variables too. Decimal numbers are sometimes referred to as floating point numbers or floats.

**Note**: when you compute numbers, they are computed with finite precision. Operations using floating points may lead to different results than the desired outcome.

# Multiply Two Decimals with JavaScript

In JavaScript, you can also perform calculations with decimal numbers, just like whole numbers.

Let's multiply two decimals together to get their product.\

Ex-let mul=2.0\*2.5;

mul=5;

# Divide One Decimal by Another with JavaScript

Now let's divide one decimal by another.

Ex-const quotient = 4.4 / 2.0;

# Finding a Remainder in JavaScript

The remainder operator % gives the remainder of the division of two numbers.

**Example**

5 % 2 = 1 because

Math.floor(5 / 2) = 2 (Quotient)

2 \* 2 = 4

5 - 4 = 1 (Remainder)

**Usage**

In mathematics, a number can be checked to be even or odd by checking the remainder of the division of the number by 2.

17 % 2 = 1 (17 is Odd)

48 % 2 = 0 (48 is Even)

Note: The remainder operator is sometimes incorrectly referred to as the modulus operator.

It is very similar to modulus, but does not work properly with negative numbers.

# Compound Assignment With Augmented Addition

In programming, it is common to use assignments to modify the contents of a variable. Remember that everything to the right of the equals sign is evaluated first, so we can say:

myVar = myVar + 5;

to add 5 to myVar. Since this is such a common pattern, there are operators which do both a mathematical operation and assignment in one step.

One such operator is the += operator.

let myVar = 1;

myVar += 5;

console.log(myVar);

6 would be displayed in the console.

# Compound Assignment With Augmented Subtraction

Like the += operator, -= subtracts a number from a variable.

myVar = myVar - 5;

will subtract 5 from myVar. This can be rewritten as:

myVar -= 5;

# Compound Assignment With Augmented Multiplication

The \*= operator multiplies a variable by a number.

myVar = myVar \* 5;

will multiply myVar by 5. This can be rewritten as:

myVar \*= 5;

# Compound Assignment With Augmented Division

The /= operator divides a variable by another number.

myVar = myVar / 5;

Will divide myVar by 5. This can be rewritten as:

myVar /= 5;

# Escaping Literal Quotes in Strings

When you are defining a string you must start and end with a single or double quote. What happens when you need a literal quote: " or ' inside of your string?

In JavaScript, you can escape a quote from considering it as an end of string quote by placing a backslash (\) in front of the quote.

const sampleStr = "Alan said, \"Peter is learning JavaScript\".";

This signals to JavaScript that the following quote is not the end of the string, but should instead appear inside the string. So if you were to print this to the console, you would get:

Alan said, "Peter is learning JavaScript".

# Escape Sequences in Strings

Quotes are not the only characters that can be escaped inside a string. There are two reasons to use escaping characters:

To allow you to use characters you may not otherwise be able to type out, such as a newline.

To allow you to represent multiple quotes in a string without JavaScript misinterpreting what you mean.

We learned this in the previous challenge.

Code Output

\' single quote

\" double quote

\\ backslash

\n newline

\t tab

\r carriage return

\b word boundary

\f form feed

Note that the backslash itself must be escaped in order to display as a backslash.

# Concatenating Strings with Plus Operator

In JavaScript, when the + operator is used with a String value, it is called the concatenation operator. You can build a new string out of other strings by concatenating them together.

Example

'My name is Alan,' + ' I concatenate.'

Note: Watch out for spaces. Concatenation does not add spaces between concatenated strings, so you'll need to add them yourself.

Example:

const ourStr = "I come first. " + "I come second.";

The string I come first. I come second. would be displayed in the console.

# Concatenating Strings with the Plus Equals Operator

We can also use the += operator to concatenate a string onto the end of an existing string variable. This can be very helpful to break a long string over several lines.

Note: Watch out for spaces. Concatenation does not add spaces between concatenated strings, so you'll need to add them yourself.

Example:

let ourStr = "I come first. ";

ourStr += "I come second.";

ourStr now has a value of the string I come first. I come second..

# Constructing Strings with Variables

Sometimes you will need to build a string. By using the concatenation operator (+), you can insert one or more variables into a string you're building.

Example:

const ourName = "freeCodeCamp";

const ourStr = "Hello, our name is " + ourName + ", how are you?";

ourStr would have a value of the string Hello, our name is freeCodeCamp, how are you?.